EXHIBIT N

Last Level Cache (LLC) Performance of Data Mining Workloads On a CMP — A Case Study of Parallel Bioinformatics Workloads

Aamer Jaleel
Intel Corporation, VSSAD
aamer.jaleel@intel.com

Matthew Mattina*
Tilera Corporation
mmattina@tilera.com

Bruce Jacob
University of Maryland, College Park
Dept. of Electrical and Computer Engineering
blj@eng.umd.edu

Abstract

With the continuing growth in the amount of genetic data, members of the bioinformatics community are developing a variety of data-mining applications to understand the data and discover meaningful information. These applications are important in defining the design and performance decisions of future high performance microprocessors. This paper presents a detailed data-sharing analysis and chip-multiprocessor (CMP) cache study of several multi-threaded data-mining bioinformatics workloads. For a CMP with a three-level cache hierarchy, we model the last-level of the cache hierarchy as either multiple private caches or a single cache shared amongst different cores of the CMP. Our experiments show that the bioinformatics workloads exhibit significant data-sharing—50–95% of the data cache is shared by the different threads of the workload. Furthermore, regardless of the amount of data cache shared, for some workloads, as many as 98% of the accesses to the last-level cache are to shared data cache lines. Additionally, the amount of data-sharing exhibited by the workloads is a function of the total cache size available—the larger the data cache the better the sharing behavior. Thus, partitioning the available last-level cache silicon area into multiple private caches can cause applications to lose their inherent data-sharing behavior. For the workloads in this study, a shared 32MB last-level cache is able to capture a tremendous amount of data-sharing and outperform a 32MB private cache configuration by several orders of magnitude. Specifically, with shared last-level caches, the bandwidth demands beyond the last-level cache can be reduced by factors of 3–625 when compared to private last-level caches.

1. Introduction

Recent trends in industry show that the future of high performance computing will be defined by the performance of multi-core processors [1, 4, 5]. Additionally, *recognition, mining, and synthesis* (RMS) workloads in the fields of medicine, investment, business and gaming are emerging as the memory intensive workloads that will run on these CMPs [21]. Within these fields, one of the most important and growing application domains is the field of bioinformatics, where workloads mine enormous amounts of genetic data to discover knowledge [14]. This motivates investigating the performance characteristics of these data-mining workloads to help define the suitable microarchitectural parameters of future CMPs.

As multi-core processors become pervasive and the number of on-die cores increases, a key design issue facing processor architects will be the hierarchy and policies for the on-die last-level cache (LLC). The most important application characteristics that drive this cache hierarchy and design are the amount and type of sharing exhibited by important multi-threaded applications. For example, if the target multi-threaded applications exhibit little or no sharing, and the threads have similar working set sizes, a simple "SMP on a chip" strategy may be the best approach. In such a case, each core has its own private cache hierarchy. Any memory block that is shared by more than one core is replicated in the hierarchies of the respective cores, thereby lowering the effective cache capacity. On a cache miss, the hierarchies of all the other cores' caches must be snooped (depending on the specifics of the inclusion policy). On the other hand, if the target multi-threaded applications exhibit a significant amount of sharing, or the threads have varying working set sizes, a shared-cache CMP is more attractive. In this case, a single, large, last-level cache—which may be centralized or distributed depending on bandwidth requirements—is shared by all the on-die cores. Cache blocks that are referenced by more than one core are not replicated in the shared cache. Furthermore, the shared cache naturally accommodates variations in the working set sizes of the different threads. In essence (and at the risk of oversimplifying), the CMP design team, building a chip with C cores and having silicon area for N bytes of last-level cache, must decide between building C private caches of N/C bytes each, or one large shared cache of N bytes. Of course, there is a large solution space between the two extremes, including replication of read-only blocks, migration of blocks, and selective exclusion to name just a few. However, the key application characteristics concerning the amount of data-sharing and the type of sharing is important for the entire design space, and thus we focus on these characteristics independently of the specific techniques used in the last-level cache. Since we have focused on the miss-analysis of shared caches on CMPs, we also ignore the impact of latency on overall performance.

Having alluded to the fact that future high-performance processors are tending towards CMPs, the question now is: *What are the important workloads of the future that will run on these CMPs?*

Recent studies have shown that the amount of data in the world is increasing by 30% each year [21]. Of the many different contributors to this growing mass of data, the biotechnology community is playing a major role via contribution of enormous amounts of genetic data

*Matthew Mattina's contribution to this work was while he was an architect at Intel Massachusetts Microprocessor Design Center (MMDC).

into GenBank, a database of all publicly known biological sequences. With the amount of genetic data more than doubling each year [14], members of the bioinformatics community have developed a variety of data-mining applications to gather meaningful information from the data and discover knowledge. With the completion of the sequencing of the human genome in 2001, the focus in bioinformatics has now shifted from gathering and sequencing data to developing intelligent algorithms that mine the massive amounts of known DNA, RNA, and protein data, with the intent of discovering previously unknown relationships, structures, and insights. These algorithms are high performance computing challenges and will help define the design and performance decisions of future high performance microprocessors.

Thus, for the purpose of this study, we chose the emerging class of bioinformatics applications as our target application set. With this in mind and with the industry transition towards CMPs, this paper makes the following contributions:

- We perform a detailed analysis of the sharing behavior of bioinformatics workloads and show that most of them exhibit a tremendous amount of data-sharing. With a shared cache, we show that 50–95% of the data cache is shared by different threads of the workload and as many as 98% of the accesses to the last-level cache are to shared cache lines. Furthermore, application sharing behavior is a function of the total size of the data cache— the larger the data cache the more an application is able to exploit its sharing behavior. For example, an application with a 4MB shared last-level data cache can exhibit 10% data-sharing; however, increasing the data cache size to 32MB shows that the workload actually exhibits 90% data-sharing. We show that for most of the workloads studied, a shared 32MB last-level cache is able to capture the bulk of an application's data-sharing. Furthermore, for these workloads, a shared 32MB last-level cache can reduce the bandwidth demands beyond the last-level cache by a factor of 3–625 when compared to a private last-level cache configuration. This implies that such workloads need the maximum cache size possible to exploit their inherent data-sharing behavior. Reducing the cache size or partitioning the cache into multiple private caches can cause a degradation in overall cache performance.
- We show that there is a direct correlation between the amount of data-sharing and the performance of shared caches. The time varying behavior of our workloads show multiple phases of execution where some phases exhibit significant data-sharing and some phases that do not. For such workloads, we observe that a shared last-level cache offers tremendous benefits during the data-sharing phase. To our knowledge, there is no prior study that correlates the data-sharing behavior of workloads with the performance of shared or private caches.
- We also investigate the impact of scaling the number of threads of the workload on cache performance. By scaling the number of workload threads from 4 to 8 and 16, and assuming CMPs of 8 and 16 cores each, we observe that shared last-level caches outperform a private last-level cache by 40-60% when comparing overall cache miss-rate.

The rest of the paper is organized in the following manner. Section 2 provides a brief description of the parallel bioinformatics workloads. Section 3 describes our methodology and the metrics used to measure application data-sharing. Section 4 provides the sharing characteristics of the workloads and presents the cache performance of private and shared caches. Section 5 presents related work. Finally in Section 6 we provide conclusions of this study.

2. Background

In this section, we provide a brief description of the OpenMP parallel bioinformatics workloads studied by Chen et al. [16]. Interested readers are referred to the original article on the description and scalability of these workloads [16].

- **GeneNet**: This application is used to measure the regulatory relationship between genes. One of the main goals of molecular biology is to understand the regulation of protein synthesis and its relation to internal and external signals. In the GeneNet application, each gene is represented as a variable of a Bayesian network, and the gene expression problem is formulated as a Bayesian network structure-learning problem. GeneNet uses hill-climbing as its main search algorithm. The algorithm is written in C++ with some details implemented using Intel's open source Probabilistic Networks Library (PNL). The training data input is the cell cycle data of Yeast (173 sequences) [16]. The total memory working set size of this application is 350MB.
- **SNP:** This application is used to measure and understand the patterns of Single Nucleotide Polymorphisms (SNPs). SNPs are a small genetic change or substitution in the nucleotides of an individual's DNA sequence. An important goal here is to understand the reasoning behind these substitutions. In the SNP application, each possible nucleotide is represented as a random variable, and all possible relations between the different nucleotides are modeled using a Bayesian network structure. Like GeneNet, the SNP application also uses hill-climbing as its main search algorithm. The algorithm is written in C++ with some details implemented using the Probabilistic Networks Library (PNL). The training input is a 30MB freely downloadable data set from the HGBASE (Human Genic Bi-Alletic Sequences), a database of SNPs [3]. There are a total of 616,179 SNPs sequences in the training data set and each sequence has a length of 50 [16]. The total memory working set size of this application is 170MB.
- **SEMPHY:** This application is a tool for constructing phylogenetic trees. Phylogenetic trees are used to represent the relationship among different species and possibly describe the course of evolution. The construction of a phylogenetic tree is a high performance computing problem, especially with the growing mass of biological data. The SEMPHY application uses the structural expectation maximization (SEM) algorithm [22] as its main search algorithm [8]. The algorithm is written in C++ and handles both DNA and protein sequences. The input data set are sequences from the Pfam database [6]. The total memory working set size of this application is 90MB.
- **Support Vector Machines Recursive Feature Elimination (SVM-RFE):** This application is used to eliminate gene redundancy from a given input data set in order to provide compact gene subsets. It uses Support Vector Machines (SVM) as the means to classify genes into different subsets. The SVM-RFE algorithm is written in C++ and uses the Intel Math Kernel Library (MKL) to enhance performance. The input data set to the application is a microarray data set involving ovarian cancer.

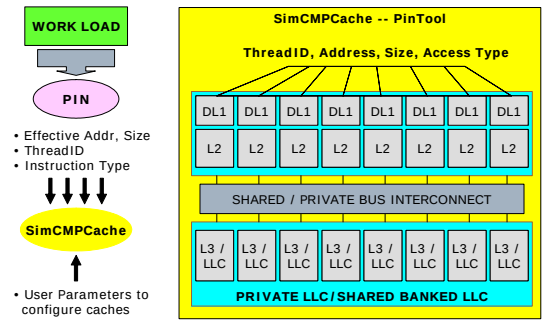


Table 1: Simulation Methodology

Instruction Cache:	Not Modeled
L1 Data Cache:	32 KB, private, write-through cache, 4-way set associative, 64B linesize, LRU replacement
L2 Data Cache:	256 KB, private, writeback cache, 8-way set associative, 64B linesize, LRU replacement
L3 / LLC Data Cache:	4, 8, 16, 32, and 64 MB, private/shared, writeback cache, 16-way set associative, 64B linesize, LRU replacement
Cache Coherence:	Invalidate-based MSI protocol
Threads Per Workload:	4, 8, and 16 Threads

Figure 1: Simulation Methodology. (a) The relationship between the workload, Pin, and the simCMPcache *pin tool* (b) An example of an 8-core configuration of simCMPcache (c) Configuration parameters for the study in this paper.

The ovarian data set contains 253 (tissue samples) x 15154 (genes) expression values. The total memory working set size of this application is 300MB.

- **Parallel Linear Space Alignment (PLSA):** This application is used to identify the similarities or differences between two genetic sequences, e.g. DNA/protein sequences. The similarity between two sequences (or the lack of it) can provide insight on understanding the functionality, structure, and evolutionary relationship of the two sequences. The application uses a dynamic programming approach to solve the sequence similarity problem, with the main algorithm being the Smith-Waterman algorithm [29]. The application is written in C++ and takes as inputs two sequences each 30,000 letters long. The total memory working set size of this application is 14MB.

3. Methodology

We now describe the tools and the hardware platform used to simulate the cache memory hierarchy.

3.1. Pin

Pin[7, 28] is a tool for the dynamic instrumentation of application binaries. It supports Linux executables for Intel® Xscale®, IA-32 (32-bit), IA-32E (64-bit), and Itanium® processors. Pin is similar to the ATOM[31] toolkit for Compaq's Tru64 Unix on Alpha processors. Like ATOM, Pin provides an infrastructure for writing program analysis tools called *pin tools*. The two main components of a Pin tool are instrumentation and analysis routines. Instrumentation routines utilize the rich API provided by Pin to insert calls to user defined analysis routines. These calls are inserted by the user at arbitrary points in the application instruction stream. Instrumentation routines are useful in defining what characteristics of an application to instrument. Analysis routines are called by the instrumentation routines at application run time. Besides instrumenting single-threaded applications, Pin also supports the instrumentation of multi-threaded applications. Pin automatically detects the creation of threads and internally creates contexts for the newly created threads without any additional user support. The scheduling of different threads of the application is controlled by the operating system.

3.2. SimCMPcache — A CMP cache simulator

For the purpose of our memory-characterization study, we implement *simCMPcache*, a pin tool that simulates the cache hierarchy of a CMP. Figure 1a provides an overview of how workload binaries, Pin, and simCMPcache interact with each other. The workload essentially runs on top of Pin; Pin captures relevant information from the workload and passes it to the simCMPcache pintool. Specifically, for every memory instruction in the workload, Pin passes to simCMPcache the associated thread ID, effective address, data size, and instruction type (load/store). SimCMPcache then takes the incoming memory instruction information and simulates cache performance using its internal cache model.

The cache model in simCMPcache is fully configurable based on parameters provided by the user. Figure 1b provides an illustrative view of simCMPcache configured as an 8-core CMP. We model a three level cache hierarchy—L1, L2, and L3 (last-level cache). The different levels of the cache can either be private or shared amongst the CMP cores. We enforce inclusion between all levels of caches. We model an MSI invalidate-based cache coherence protocol where the states for the cache line are *Modified*, *Shared*, and *Invalid*. On a write request, invalidates are sent to the relevant private caches to invalidate any matching entries. Similarly, when read requests miss in the private caches, remote dirty lines (if any) are required to perform a write-back to lower levels of memory before servicing the miss.

SimCMPcache is capable of gathering a variety of statistics. On a per-application-thread basis, the simulator tracks the application instruction profile, cache statistics in terms of accesses and misses, sharing characteristics of the last-level cache, and statistics on the coherence traffic between caches. These statistics can be written to a logfile when the program has finished execution. However, to characterize the time varying behavior of the application, cache statistics are logged to file every 10 million instructions committed by any thread of the application.

Table 1 presents the methodology for studying the cache performance of our workloads. For the purpose of this study, we assume a perfect instruction cache. The L1 data cache is 32KB in size, 4-way set associative, with 64-byte linesize and a write-through policy. The L2 cache is 256KB in size, 8-way set associative, with 64-byte linesize and write-back policy. For the purpose of this study we do not consider changing the parameters of the L1 and L2 caches. Finally, the last-level cache is either 4/8/16/32/64 MB, 16-way set

Table 2: Dynamic Application Instruction Distribution

	Instruction Count (Billions)	%Memory Instructions	% ALU Instructions	%Memory Read Instructions
PLSA	418.53 B	85.05%	14.95%	49.40%
GeneNet	1,491.49 B	65.54%	34.46%	48.89%
SEMPHY	811.96 B	61.15%	38.85%	46.33%
SNP	59.59 B	45.19%	54.81%	36.85%
SVM	40.91 B	43.52%	56.48%	38.75%

Table 3: First and Second Level Cache Statistics

	DL1 Accesses / 1000 Inst	DL1 Misses / 1000 Inst	DL1 Miss-rate	DL2 Misses / 1000 Inst	DL2 Read Misses / 1000 Inst	DL2 Miss-rate
PLSA	850	0.85	0.10	0.04	0.02	0.01
GeneNet	656	4.50	0.68	3.73	3.55	2.21
SEMPHY	611	7.25	1.18	6.83	3.96	4.49
SNP	452	11.57	2.56	11.57	11.33	10.50
SVM	435	97.20	22.30	51.90	50.34	48.05

associative, with 64-byte linesize and write-back policy. All caches allocate on a store miss and use the LRU cache-line replacement policy. The L1 and L2 cache are modeled as private caches and the last-level cache can either be private or shared.

3.3. Hardware platform

To capture the memory-access characteristics of multi-threaded workloads, our experiments are run on 4-way or 8-way shared memory multi-processor (SMP) systems of Intel® Pentium® 4 processors. The systems all run the RedHat Linux 7.1 operating system with hyper-threading enabled. Such systems allow us to run our workloads with 1 to 16 threads. The workloads are compiled using the *icc* compiler with optimization flags -O3. For the purpose of this study our workloads are executed with 4, 8, and 16 threads.

3.4. Metrics

To understand the sharing behavior of parallel workloads, we define the following metrics to measure the degree of data-sharing existent in parallel applications.

- **Shared Cache Line:** Cache lines in a shared data cache can either be private or shared. When executing parallel applications, an important workload characteristic is a measure of how much data is shared between threads. One way of measuring the degree of sharing is to measure the number of cache lines that are touched by different cores of a CMP. Assuming that the threads of an application execute on different cores of a CMP, and threads do not migrate between cores, we define a *shared cache line* as one that is accessed by more than one core of a CMP during its *lifetime* in the cache. For the purpose of this study, we classify a shared cacheline as either read-only shared or read-write shared. Read-only shared cache lines are those cache lines that are only read from and not written to. For example, while searching a database in parallel, more than one thread of a workload can read the same cache line. A read-write shared cache line is a shared cache line that is used as a means of communication between threads. A producer thread writes to a cache line and a consumer reads data from the same cache line. The shared cache line metric is useful in determining the shared data footprint of a workload and the type of sharing the workload exhibits.
- **Shared Access:** An access to a shared cache line is defined as a *shared access*. This metric is useful in determining the variation and frequency of accesses to shared or private cache lines. Such a metric not only indicates the amount of data-sharing prevalent in an application, but it also provides intuition for the choice of cache design for parallel applications. For example, if most of the cache accesses are to shared data, then perhaps shared caches may provide better cache performance than private caches.
- **Active-Shared Access:** An active shared access is an access to a shared cache line with the condition that the last core that accessed the shared cache line is *not* the same as the current core. For example, if the accesses to a shared cache line is represented by the following core ids: ...1, 2, 2, 2, 1, 3, 4, 3, 2, 2, 2, 3, 2..., the accesses by the underlined core IDs are *active-shared accesses*. Such a metric is useful in identifying and characterizing whether workloads share cache lines interactively or in a serial fashion. Such information can be useful for determining the benefits of data migration to cache banks closer to the accessing cores.

4. Characterization results

The OpenMP implementation of the bioinformatics workloads in this study typically start with a serial initialization or training phase where only the main thread is active. After the serial phase, threads are created to perform work during the parallel phase. Even though data is gathered over the entire run of each workload, unless otherwise mentioned, we present the behavior of the workloads during the parallel phase of execution, which dominates overall execution time. The data presented is averaged over the periodic logs generated by our cache simulator.

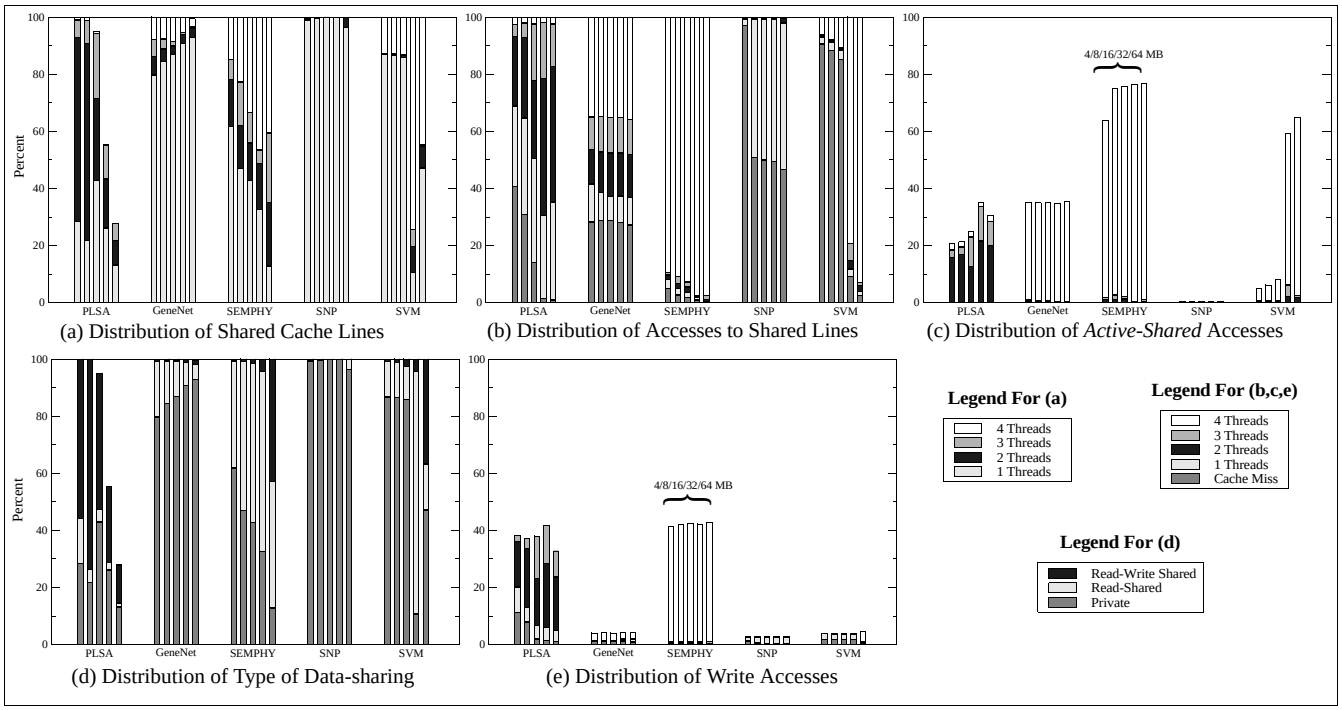


Figure 2: Sharing Behavior with a Shared Last-Level Cache. Distribution of (a) shared data in last-level cache (b) cache accesses to data in last-level cache (c) cache accesses to actively shared data in last-level cache (d) classification of data-sharing in last level cache (e) write accesses to data in last-level cache.

4.1. Application instruction profile

Table 2 presents the dynamic instruction profile for the different workloads when run with 4 threads. For each application we present the total number of instructions executed across all threads and the distribution of instructions categorized into memory instructions and ALU instructions. With the CISC nature of the x86 ISA, instructions can perform arithmetic or logic computation based on operands that reside either in memory or the register file. Accordingly, we define memory instructions as those instructions that have one or more operands in memory and ALU instructions as those instructions that have all their operands in the register file. We also provide the breakdown of total instructions that were memory read instructions.

Based on the application instruction profile, we observe that the workloads consist of roughly 43–65% memory instructions, with as many as 85% for the PLSA workload. We also observe that memory read instructions constitute 60–90% of total memory instructions. The large share of memory instructions, especially memory read instructions, is to be expected as these workloads work through large amounts of data in attempts to discover meaningful patterns or relationships between data.

4.2. Workload L1/L2 cache behavior

Table 3 presents the L1 and L2 cache statistics for the different workloads. For each level of the cache, we present the number of accesses and misses per 1000 instructions (committed) as well as the overall cache miss-rate. From the table, PLSA has the lowest L1 data cache miss-rate and SVM has the largest L1 and L2 data cache miss-rates. With the exception of PLSA and SVM, comparing the L1 and L2 misses per 1000 reveals that 80–95% of read accesses that miss in the L1 data cache usually also miss in the L2 data cache. This implies that these workloads have two different data sets: one that is small and frequently used and another that is large and does not fit into the L2 data cache.

4.3. Cache utilization and data-sharing

We now present the workload cache utilization as well as data-sharing behavior for five last-level cache sizes: 4/8/16/32/64 MB. With four threads per workload, Figure 2a illustrates, on the y-axis, the percent of cache utilized as well as the distribution of cache lines shared amongst different threads. In the figure, for each workload, the five bars represent the different last-level cache sizes in increasing order with the left most bar representing the 4MB last-level cache. Each bar is split into four categories: the bottommost portion represents those cache lines that are private, the next one up represents those cache lines shared by 2 threads, the next one up represents those cache lines shared by 3 threads, and finally the topmost portion represents those cache lines that are shared by 4 threads. From the figure, with the exception of PLSA, these workloads fully utilize a 64MB last-level cache (last bar graph for each workload). PLSA's entire memory foot print fits in a 16MB last-level cache. Unlike SPEC workloads (that barely utilize a 2-8MB last-level cache), the large memory working-set sizes of these workloads will continue to put pressure on processor and DRAM architects to reduce the ever growing "memory gap".

Figure 2a also illustrates that the workloads present a varying amount of data-sharing. Some workloads exhibit very little data-sharing, as in the case of SNP where only 2% of a 64MB cache is shared by two threads. GeneNet, SEMPHY, and SVM demonstrate a cache with data that is either private or shared amongst two to four threads. On the other hand, PLSA demonstrates data that is either private or shared amongst two or three threads. This behavior goes to

show that even though workloads are run with four threads, workloads need not share data cache lines amongst all four threads. For example, based on PLSA's dynamic programming algorithm, data can only be shared by a maximum of three threads. However, there is one caveat: even though algorithmically data can not be shared by all threads of a parallel workload, data-sharing between all threads may still exist for purposes of synchronizing between all the threads of the workload.

Based on Figure 2a, it can be seen that the amount of sharing varies with the size of the data cache. Increasing the size of the data cache can either increase or decrease the percent of shared data in the cache. A decrease in the percent of shared data with increasing cache sizes implies that an application's shared data footprint is smaller than the private data footprint. For example, in GeneNet, increasing the data cache size beyond 4MB causes the amount of shared data in the cache to decrease from 20% in a 4MB cache to 5% in a 64MB cache. Similarly, increasing the cache size beyond 32MB for SVM causes the amount of shared data in the cache to decrease from 90% in a 32MB cache to 55% in a 64MB cache. Such behavior provides a good indication of the total shared working-set of these workloads.

Alternatively, increasing the size of the cache can also increase the ratio of shared data in the cache. This is because conflict and capacity misses in smaller caches can result in the eviction of potential shared data. For example, a cache line that is actually shared amongst four cores of a CMP can be evicted due to a cacheline conflict immediately after the first core brings the cacheline into the cache. Increasing the cache capacity reduces the number of conflict misses and provides opportunity for both private and shared data to co-exist in the cache. This behavior is evident in the workloads PLSA, SEMPHY, SNP, and SVM and can be better explained by the distribution of accesses to the shared cache.

Figure 2b illustrates, on the y-axis, the distribution of accesses to the shared last-level cache. In the figure, the five bars represent the different last-level cache sizes. Each bar graph is split into five categories based on the type of cache access. Cache accesses are divided into accesses that miss in the data cache and accesses that are either to private or shared (by two, three, or four threads) cache lines. From the figure, for all workloads besides GeneNet, increasing the data cache size reduces the percent of cache misses in the last-level cache. For these workloads, a direct correlation exists between the reduction in cache misses and the increase in the amount of shared data in the cache. Particularly with SVM, almost 90% of cache accesses in a 4 or 8 MB last-level cache result in a cache miss. However, reducing the number of conflict misses by increasing the data cache size reveals that SVM, which initially exhibited very little data-sharing with a 4MB cache, *actually* exhibits a tremendous amount of data-sharing with a larger cache size (compare with Figure 2a). Based on this behavior, we can conclude that workloads require the maximum possible cache space available to exploit their inherent data-sharing behavior. We will show that attempting to reduce the size of the cache or partition the cache into multiple smaller independent caches proves detrimental to cache performance.

Figure 2b shows an interesting behavior in terms of the cache access patterns for several of the workloads. For workloads that share their data cache lines, most of the cache accesses to the last-level cache are to the shared data. For example, with a 64MB shared last-level cache, 62% of PLSA's cache accesses (in some phases of execution as much as 100%) are to its 8MB shared footprint, 60% of GeneNet's cache accesses are to its 4MB shared footprint, and 98% of SEMPHY and SVM's cache accesses are to their 56MB and 32MB shared footprints respectively. Furthermore, Figure 2c shows, on the y-axis, the distribution of active-shared accesses to the last-level cache. Recall that an *active-shared* access is an access to a shared data cache line by core C_i and the last access to the same cache line was not C_i. For the workloads, 30–80% of cache accesses are to data shared that is interactively shared by two to four threads. Thus, from Figure 2b and 2c, we conclude that workloads not only access shared data frequently, but they also access the shared data interactively rather than in a per-thread serial fashion.

Based on the data presented in this section, we observe that most of the workloads exhibit a significant amount of data-sharing. Sharing is not only exhibited by the existence of shared data in the cache but also by the large distribution of interactive accesses to the shared data. We show that sharing is exposed when other factors such as cache misses are removed from the scene. Hence, reducing or partitioning last-level caches can cause an application to lose its sharing behavior. We show later in the paper that the loss of sharing can place unnecessary demands for bandwidth on the memory subsystem, or could require extensive last-level cache snoop bandwidth if such an implementation technique is employed.

4.4. Data-sharing classification

Having demonstrated that these applications exhibit large amounts of data-sharing, we now analyze the type of data-sharing. Figure 2d illustrates, on the y-axis, the distribution of cache lines categorized into those that are private, read-only shared, and read-write shared. From the figure, 30–50% of PLSA, SEMPHY, and SVM's cache lines are read-write shared while GeneNet and SNP's cache lines are mostly private. To better understand the type of data-sharing in these workloads, Figure 2e illustrates, on the y-axis, the percent of write accesses to the shared last-level cache. From the figure, GeneNet, SNP, and SVM exhibit negligible write-accesses, hence we can classify them as read-shared workloads. Even though SVM exhibits 30% read-write shared cache lines, the negligible amount of write-accesses to the last-level cache classifies SVM as a read-shared workload. On the other hand, roughly 30–40% of SEMPHY and PLSA's write accesses are to shared data. This implies that both PLSA and SEMPHY are read-write shared workloads. This motivates investigation into the coherence traffic behavior of these workloads. Based on the type of data sharing exhibited, we conclude that even though some workloads exhibit negligible write accesses to the shared last-level cache, the fact that the workloads exhibit extensive read sharing emphasizes the need for shared caches to avoid unnecessary duplication of data.

4.5. Performance of private and shared last-level caches

Figure 3 presents the cache metrics for the private and shared last-level cache configurations for the different workloads. In each figure, the first five bars represent private last-level caches, and the last five bars represent shared last-level caches. The performance of the private cache is plotted as the average of the performance of all private caches. We remind the reader that when using private caches, the total on-die last-level cache is partitioned equally amongst different cores of the CMP. Thus, with 4 cores, and with on-die last-level cache sizes of 4/8/16/32/64 MB, the private last-level cache sizes are 1/2/4/8/16 MB each. For each cache, we present accesses per

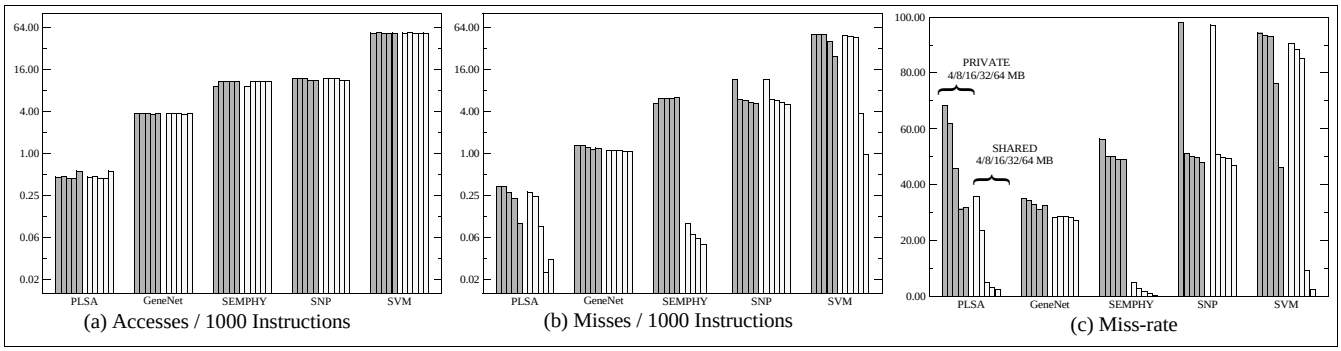


Figure 3: Performance of Private and Shared Last-Level Caches. (a) Accesses / 1000 Instructions (b) Misses / 1000 Instructions (c) Miss-rate. Note that figures (a) and (b) are on a logarithmic scale to accommodate the varying behavior of the different workloads.

1000 instructions, misses per 1000 instructions, and cache miss-rate. The accesses and misses per 1000 are presented on a logarithmic scale to accommodate the varying behavior of the workloads.

As expected, from Figure 3a, the cache access rates for both the private and shared last-level cache are identical. Furthermore, varying the last-level cache size does not affect the cache access rate. This is to be expected as we do not vary the sizes of the first and second level caches. We note that minor variations are expected, and are due to the already known repeatability problem [13] of running workloads multiple times on real machines with real operating systems.

As expected, Figure 3c shows that increasing the size of the private last-level cache aids in reducing the overall cache miss-rate by 3–50%. GeneNet and SNP show no significant improvements in cache performance with larger private (or shared) caches, most likely due to little data reuse and frequent misses in the last-level cache (one-third to one-half of accesses to the last-level cache result in cache misses). We observe that all workloads, besides PLSA, require heavy memory bandwidth, with 4–50 misses per 1000 instructions (GeneNet has execution phases with 4–5 misses per 1000 instructions). PLSA fits well in the private L2 cache as the core of the dynamic programming algorithm works on small blocks of a matrix before moving onto the next block.

We now compare the performance of shared last-level caches to that of private caches. From figure 3b, we observe that with small shared last-level caches, a shared last-level cache in general performs as well or better than the same sized partitioned private last-level cache. For example, a shared 8 MB last-level cache has similar cache performance as a 32 MB private last-level cache configuration (i.e. each of the four cores has an allotted private 8MB cache). This can be explained by the fact that accesses to shared data causes only one miss in a shared cache. However, with a private cache configuration, an access to uncached shared data results in a cache miss in the private cache of each core. For example, in a four-core CMP, an access to data (that is initially not present in the last-level cache) shared by all four cores results in a 100% overall cache miss-rate with the use of a private cache and a 25% overall cache miss-rate with the use of a shared cache. This is because, with a shared cache, the access by the first core fills the cache with the missing data, hence all successive requests for the same data from other cores of the CMP hit in the cache. On the other hand, with a private cache, all cores miss in their respective private caches.

We observe that the reduction in miss-rate translates into reductions in the bandwidth demands beyond the last-level cache by as much as a factor of 625. Based on Figure 3b, the number of last-level cache misses (per 1000 instructions) for a 64MB cache reduces from 0.1 to 0.03 for PLSA, 6.27 to 0.01 for SEMPHY and 24.2 to 0.98 for SVM when the last-level cache is changed from private to shared. Furthermore, these workloads benefit the most from a shared last-level cache as 98% of their last-level cache accesses were to shared data. This reinforces the fact that workloads that frequently access shared data tend to benefit the most from shared caches.

From the figure, we also observe that GeneNet and SNP receive little or no benefit with a shared last-level cache. Both these workloads exhibit little data-sharing and have poor last-level cache performance. For the different last-level cache sizes that we simulated, independent of the size of the last-level cache, roughly 30–50% of all accesses result in a cache miss (see Figure 2b). This is perhaps the primary reason for these workloads not exhibiting significant data sharing—potential shared cache lines are evicted from the cache due to conflict misses before other threads have a chance to access them. Based on such behavior, data-sharing parameters may need to be considered in cache allocation and eviction policies. This is discussed as part of our future work.

PLSA, SEMPHY, SNP, and SVM all present a constant behavior during the course of execution—they either share or do not share their data. However, workloads can have multiple phases of execution where some phases exhibit significant data-sharing while other phases exhibit a lesser degree of data-sharing. For such workloads, a shared last-level cache can offer tremendous benefits in the data-sharing phase. Of the five workloads, GeneNet displays this behavior. Figure 5b (in the appendix) shows the detailed cache behavior of GeneNet over its entire run. From the figure, during the first 1/8th phase of execution, GeneNet exhibits a tremendous amount of data-sharing—close to 80% of cache accesses are to shared data. During this phase, a shared last-level cache has roughly one-third the miss-rate of a private last-level cache (22% vs. 78%). For other phases of execution, 50–60% of cache accesses are to shared cache lines, and during this phase the shared last-level cache performs better than the private last-level cache by 5–10% on average. Thus, the time-varying sharing behavior of GeneNet further reinforces the fact that a shared last-level cache is highly beneficial when workloads exhibit a large amount of data-sharing.

Based on the data presented, we conclude that workloads or phases of execution that heavily access shared data tend to benefit the most with shared last-level caches. Since the workloads in this study display a large amount of data-sharing, a shared last-level cache can provide tremendous opportunity to reduce the bandwidth demands beyond the last-level cache. Furthermore, with parallel applications in

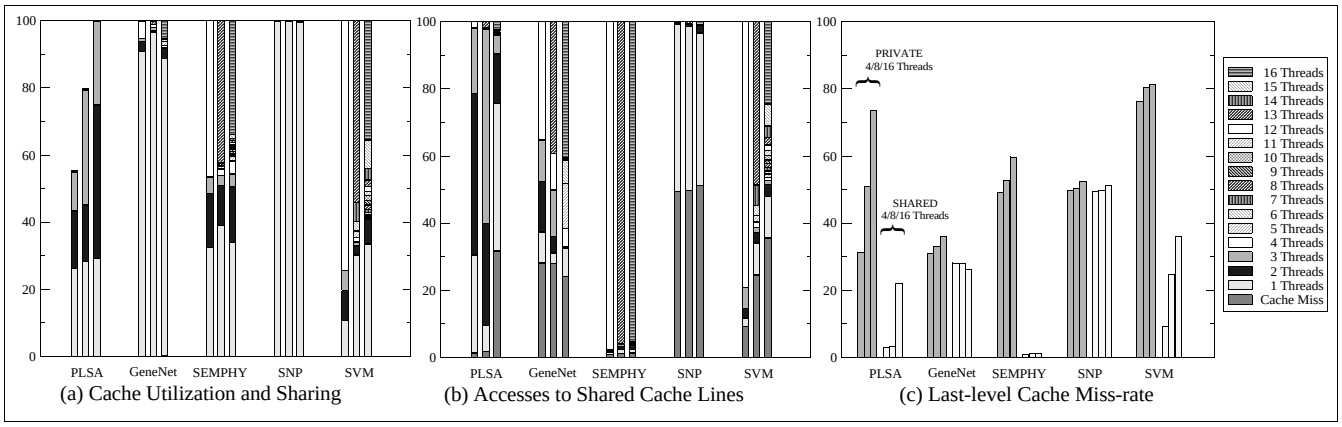


Figure 4: Cache Performance of Workloads Executed With 4, 8, and 16 Threads With a 32MB Last-Level Cache. (a) Distribution of data in shared last-level cache (b) Distribution of accesses to data in shared last-level cache (c) Performance comparison of private and shared last-level caches.

other emerging domains [9, 12], extensive data-sharing may force shared last-level caches to be a necessity in future CMPs.

4.6. Performance of private and shared last-level caches in larger CMPs (8 and 16 cores)

Based on the performance of the different caches, we observe that maximum cache performance is achieved while moving from a 16MB cache to a 32MB cache. Beyond a 32MB cache size, all workloads, other than SVM, receive marginal improvements in terms of cache performance. Based on this data, we chose to determine the cache performance of these workloads with a 32MB last-level cache while varying the core count from 4 to 8 and 16 cores (i.e. the workloads are run with 8 and 16 threads). Note that our purpose for scaling the workloads is to quantify the impact on miss-rate with an increasing core count while keeping the cache size fixed. Figure 4 illustrates the amount of data-sharing, distribution of cache accesses, and a comparison of private and last-level cache performance for the different workloads. In Figures 4a and 4b, the three bar graphs for each workload represent the 4, 8, and 16 threaded runs. The legend of each individual bar graph is similar to the legends of Figure 2b except that additional legend entries are present for the 8 and 16 thread configurations to represent cache lines that are shared amongst 5, 6, 7,..., 14, 15, and 16 threads.

Based on Figure 4a and 4c, scaling the number of threads for the workloads increases the cache utilization as well as the cache miss-rate. The increase in cache utilization can be explained by the fact that the working sets of these workloads are not entirely shared. Each additional worker thread adds its own private data to the workload data footprint, hence increasing the overall data footprint of the workload. Consequently, the increase in footprint translates into an increase in the overall cache miss-rate. Furthermore, we also point out that an increase in cache miss-rate with private caches can also be explained by the fact that the size of the private partitions of each core decreases when the number of on-chip cores increases. For example, with a 32MB last-level cache, each private cache in a 4-core CMP is 8MB in size, while in a 8 and 16-core CMP each private cache is 4MB and 2MB in size respectively. Thus, the reductions in the sizes of the private cache partitions per core also contributes to an increase in the overall cache miss-rate with private last-level caches.

Figure 4c compares the cache performance of private and shared caches. In the figure, the first three bars represent the workloads run with 4, 8 and 16 threads with a private cache, and the last three represent those that are run with shared caches. For the workloads in this study, with larger CMPs a shared cache configuration reduces the overall miss-rate by 40–60% when compared to a private cache configuration. Additionally, the performance of shared caches varies as the core count of the CMP is scaled up to 8 and 16 cores. PLSA and SVM experience a 20–30% increase in the over-all cache miss-rate while GeneNet, SNP, and SEMPHY experience marginal increases in the over-all cache miss-rate. With 4 threads, SVM and PLSA fit well into a 32MB cache, however scaling them to 8 and 16 threads increases the cache miss-rate. This can be explained by the fact that both PLSA and SVM's data footprint is not entirely shared, each new thread adds additional private data footprint. As a result, Figure 4b illustrates that the additional per-thread private data footprint causes as many as 30% of cache accesses to miss in the data cache. The increase in the number of cache misses results in potential shared lines to be evicted from the data cache hence reducing the total amount of shared data in the last-level cache. On the other hand, for the workloads GeneNet and SEMPHY, the additional per-thread private data portion of the total working set is relatively small, hence the marginal increase in the over-all cache miss-rate. This implies that such workloads can potentially achieve *super-linear speedup* with the scaling up of the number of threads as cache performance is unaffected.

Based on the data presented in this paper, the key point is that shared last-level caches are essential for the workloads in this study to have good cache performance. By scaling the number of CMP cores, we conclude that the performance of a shared last-level cache can outperform a private last-level cache by 40–60% in terms of over-all cache miss-rate. Hence, given the option of designing private or shared last-level caches, we conclude that shared last-level caches are a necessity for this emerging class of data-mining workloads.

5. Related work

Bioinformatics has emerged as an important application domain for future high performance microprocessors. Consequently, industry has invested resources in characterizing the scalability and performance of common bioinformatics applications. Cheng et al. looked at the

scalability of bioinformatics applications like BLAST, FASTA, HMMER, etc. on the IBM eServer pSeries 690 [18]. Chen et al. looked at the scalability and performance of data mining bioinformatics applications [16, 17]. Sun Microsystems presented a whitepaper on the challenges faced and the opportunities available in the field of computational biology [10]. Furthermore, the Informatics Benchmarking Toolkit (IBT) provided by *BioTeam* compares the performance of common bioinformatics applications [2]. In academia, Albayraktaroglu et al. compiled the BioBench suite of bioinformatics applications and characterized their behavior and differences from the SPEC benchmark suite [14].

Recent studies have investigated the design for the cache-hierarchy of CMPs. Liu et al. discussed the tradeoffs of implementing shared and private caches and proposed a mechanism of allocating multiple last-level private caches to one core of a CMP [25]. Chishti et al. presented novel mechanisms of optimizing replication, coherence communication, and exploiting unused cache space in CMPs [19]. Zhang et al. proposed victim replication to achieve the benefits of private caches with shared caches [33]. Speight et al. looked at CMP performance through intelligently handling write-backs [30].

Characterizing the memory behavior and performance of parallel workloads is an area of ongoing research. Abandah et al. proposed a configuration-independent approach to analyze the working set, concurrency, and communication patterns, as well as sharing behavior of shared memory applications [11]. They present the *Shared-Memory Application Instrumentation Tool* (SMAIT) to measure different sharing characteristics of the NAS shared-memory applications [12]. Barroso et al. characterized the memory system behavior of commercial workloads such as Oracle, TPC-B, TPC-D, and the AltaVista search engine [15]. They performed their characterization of the memory system behavior using ATOM [31], performance counters on an Alpha 21164, and the SimOS simulation environment. Woo et al. characterized several aspects of the SPLASH-2 benchmark suite [32]. They used an execution-driven simulation with the Tango Lite [23] tracing tool. Perl et al. studied Windows NT applications on Alpha PCs and characterized application bandwidth requirements, memory access patterns, and application sensitivity to cache size [27]. Chodneker et al. analyzed the time distribution and locality of communication events in some message-passing and shared-memory applications [20]. Nurvitadhi et al. used an FPGA-based cache model (PHA$E) that connects directly to the front-side bus to analyze the shared vs. private L3 cache behavior of SPECjAppServer and TPC-C [26].

Our work differs from prior work in that it performs a detailed last-level cache characterization of the emerging class of bioinformatics data-mining workloads. We perform a detailed data-sharing analysis of these workloads over their entire execution without using any tracing mechanisms, performance counters, or “bus sniffers”. Contrary to existing trace driven and execution driven methodologies, we use the binary instrumentation approach to characterize the cache performance of workloads over their entire run. We believe this to be the first study that characterizes the cache performance of data-mining workloads from the perspective of a CMP. Additionally, we also believe this to be the first study that clearly correlates cache sizes and cache performance with workload-sharing behavior.

6. Conclusions and future work

The study in this paper gives important insight into workloads that will be very important in future high-performance machines, and this insight is valuable to architects of new CMPs. We show that the bioinformatics data-mining workloads used in this study exhibit a tremendous amount of data-sharing—50–95% of the data cache is shared by different threads of the workload. Furthermore, regardless of the amount of data cache shared, for some workloads, as many as 98% of the accesses to the last-level cache can be to the shared data. Additionally, the amount of data-sharing exhibited by the workloads is a function of the total cache size available; the larger the data cache the better the sharing behavior. Thus, rather than partitioning the last-level cache into multiple private caches, we show that a shared last-level cache is important for improving the performance of these workloads on future high performance machines. With a shared last-level cache, the bandwidth demands beyond the last-level cache can be reduced by factors of 3–625 when compared to a private last-level cache configuration. Thus, we conclude that, given the option of designing shared or private last-level caches in future CMPs, a shared last-level cache is the logical choice of implementation as it allows workloads to exploit their data-sharing behavior and significantly reduce the bandwidth demands beyond the last-level cache.

Looking ahead, our on-going work focuses on investigating the performance bottlenecks of these workloads using performance models. Additionally, we are also studying the sensitivity of the sharing behavior of these workloads to varying data input sizes. Furthermore, we are also exploring the use of novel cache allocation and replacement policies that can allow applications to exhibit the sharing behavior observed in large last-level caches with smaller last-level caches.

Acknowledgements

The authors would like to acknowledge the following individuals for their contribution to this work: Robert Cohn and C. K. Luk for their help in developing the simCMPcache pin tool, Carole Dulong for her support with the bioinformatics applications, and Joel Emer, Brinda Ganesh, Srilatha Manne, Moinuddin Qureshi, Jason Papadapolous, and Simon Steely for reviewing the paper. We would also like to thank the reviewers for their comments and suggestions.

References

[1] AMD MultiCore Technology: http://multicore.amd.com

[2] The BioTeam: Infomatics Benchmarking Toolkit (IBT), AMD Funded, http://bioteam.net/ibt/

[3] HGBase: http://hgvbase.cgb.ki.se/

[4] IBM Cell Processor: http://www-306.ibm.com/chips/techlib/techlib.nsf/products/Cell

[5] Intel Pentium Processor Extreme Edition: http://www.intel.com/products/processor/pentiumXE/index.htm

[6] PFam Database: http://www.sanger.ac.uk/Software/Pfam/

[7] PIN home page: http://rogue.colorado.edu/Pin/.

[8] SEMPHY Download Page: http://www.cs.huji.ac.il/~nir/SEMPHY/

[9] SPECOMP2001 http://www.spec.org/

[10] "Sun Briefing In Computational Biology", White Paper, December, 2003.

[11] G. A. Abandah, and E. S. Davidson. "Configuration Independent Analysis for Characterizing Shared-Memory Applications." In *Proceedings of the 12th. International Parallel Processing Symposium (IPPS)*, Orlando, Florida, 1998.

[12] G. A. Abandah. "Characterizing Shared-Memory Applications: A Case Study of the NAS Parallel Benchmarks." Technical Report, HPL-97-24, Hewlett Packard.

[13] A. R. Alameldeen and D. A. Wood. "Variability in Architectural Simulations of Multi-threaded Workloads." In *Proceedings of the 3rd International Conference on High Performance Computer Architecture (HPCA)*, Anaheim, California, 2003.

[14] K. Albayraktaroglu, A. Jaleel, X. Wu, M. Franklin, B. Jacob, C. Tseng, and D. Yeung, "BioBench: A Benchmark Suite of Bioinformatics Applications." In *the 5th International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Austin, Texas, 2005.

[15] L. A. Barroso, K. Gharachorloo, and E. Bugnion. "Memory System Characterization of Commercial Workloads." In *Proceedings of the 25t International Symposium on Computer Architecture (ISCA)*, Barcelona, Spain, 1998.

[16] Y. Chen, Q. Diao, C. Dulong, C. Lai, W. Hu, E. Li, W. Li, T. Wang, and Y. Zhang. "Performance Scalability of Data-Mining Workloads in Bioinformatics." Intel Technology Journal, Volume 09, Issue 02, May 19, 2005.

[17] Y. Chen, Q. Diao, C. Dulong, C. Lai, W. Hu, E. Li, W. Li, T. Wang, and Y. Zhang. "Performance Scalability of Data-Mining Workloads in Bioinformatics." Technical Report.

[18] Y. Cheng, J. Mark, C. Skawratananond, and T. K. Tzeng. "Scalability Comparison of Bioinformatics for Applications on AIX and Linux on IBM eServer pSeries 690." Redbooks Paper, August 2004.

[19] Z. Chishti, M. Powell, and T. N. Vijaykumar. "Optimizing Replication, Communication, and Capacity Allocation in CMPs." In *Proceedings of the 32nd International Symposium on Computer Architecture (ISCA)*, Wisconsin, Madison, 2005.

[20] S. Chodnekar, V. Srinivasan, A. Vaidya, A. Sivasubramaniam, and C. Das. "Towards a Communication Characterization Methodology for Parallel Applications." In *Proceedings of the 3rd International Conference on High Performance Computer Architecture (HPCA)*, San Antonio, Texas, 1997.

[21] P. Dubey. "Recognition, Mining and Synthesis Moves Computers to the Era of Tera." Intel Technology Journal, February 2005.

[22] N. Friedman, M. Ninio, I. Pe'er, and T. Pupko. "A Structural EM Algorithm for Phylogenetic Inference". In *Journal of Computational Biology*, 2001.

[23] S. Goldschmidt and J. Hennessey. "The Accuracy of Trace-Driven Simulations of Multiprocessors." Tech Rep. CSL-TR-92-546, Stanford University, Sept. 1992.

[24] N. Friedman, M. Linial, I. Nachman, and D. Pe'er. "Using Bayesian Networks to Analyze Expression Data." Journal of Computational Biology, 7:601-620, 2000.

[25] C. Liu, A. Sivasubramaniam, M. Kandemir. "Organizing the Last Line of Defense before Hitting the Memory Wall for CMPs." In *Proceedings of the 6th International Conference on High Performance Computer Architecture (HPCA)*, Madrid, Spain, 2004.

[26] E. Nurvitadhi, N. Chalainanont, and S. L. Lu. "Characterization of L3 Cache Behavior of SPECjAppServer2002 and TPC-C." In *Proceedings of the 19th International Conference on Supercomputing (ICS)*, Boston, Massachusetts, 2005.

[27] S. E. Perl and R. L. Sites. "Studies of Windows NT performance using dynamic execution traces." In *Proceedings of the 2nd International Symposium on Operation Systems Design and Implementation (OSDI)*, Seattle, Washington, 1996.

[28] V. Reddi, A. M.Settle, D. A. Connors and R. S. Cohn. "Pin: A Binary Instrumentation Tool for Computer Architecture Research and Education." In *Proceedings of the Workshop on Computer Architecture Education*, June 2004.

[29] T. F. Smith and Michael S. Waterman. "Identification of Common Molecular Subsequences." In *Proceedings of the Journal of Molecular Biology*, 147:195-197, 1981.

[30] E. Speight, H. Shafi, L. Zhang, R. Rajamony. "Adaptive Mechanisms and Policies for Managing Cache Hierarchies in CMPs." In *Proceedings of the 32nd International Symposium on Computer Architecture (ISCA)*, Wisconsin, Madison, 2005.

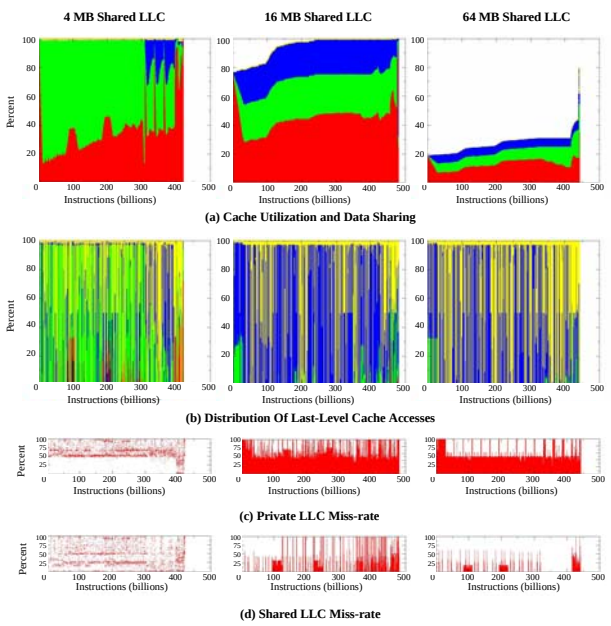
[31] A. Srivastava and A. Eustace, "ATOM: A System for Building Customized Program Analysis Tools", Programming Language Design and Implementation (PLDI), 1994, pp. 196-205.

[32] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. "The SPLASH-2 Programs: Characterization and Methodology Considerations." In *Proceedings of the 22nd International Symposium on Computer Architecture (ISCA)*, Santa Margherita Ligure, Italy, 1995.

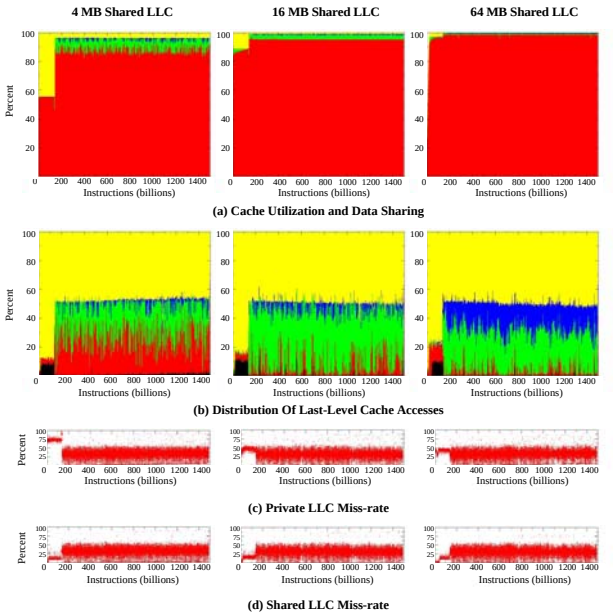
[33] M. Zhang and K. Asanovic. "Victim Replication: Maximizing capacity while Hiding Wire Delay in Tiled CMPs." In *Proceedings of the 32nd International Symposium on Computer Architecture (ISCA)*, Wisconsin, Madison, 2005.

Appendix

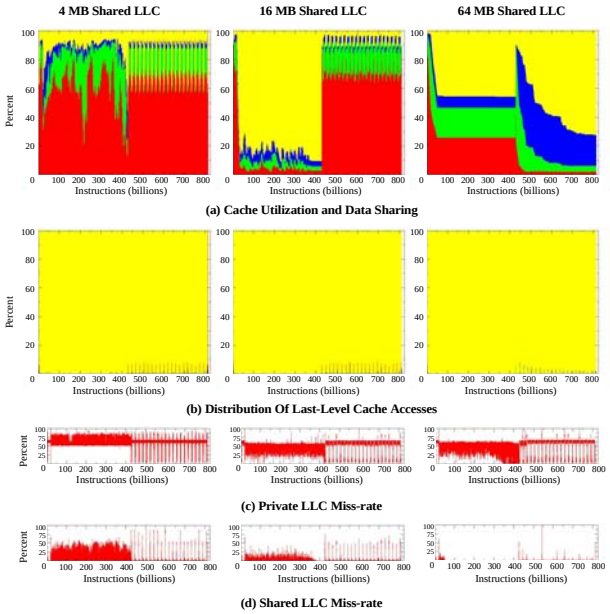
This section presents the time varying behavior of the workloads. We present the distribution of private and shared data in the cache, the distribution of accesses to private and shared data, and the private and shared last-level cache miss-rates. The data is presented for the 4MB, 16MB, and 64MB last-level cache sizes. Each graph illustrates the total instruction count (in billions) on the x-axis and the appropriate metric represented as a percent on the y-axis.



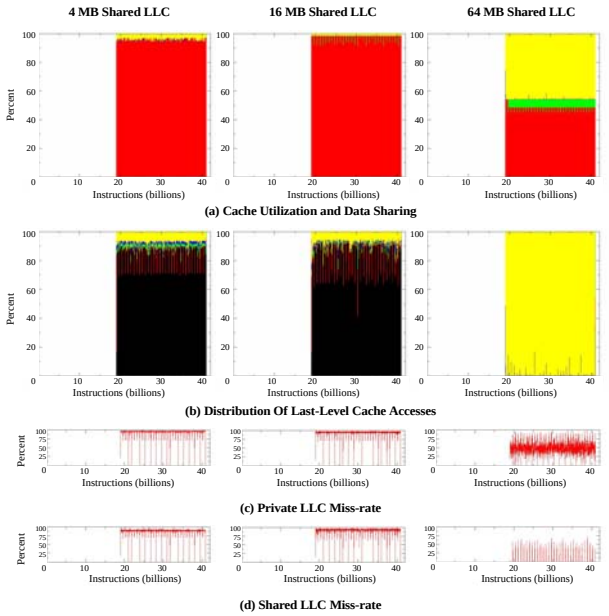
(a) LLC Behavior of PLSA With 4 Threads



(b) LLC Behavior of GeneNet With 4 Threads



(c) LLC Behavior of SEMPHY With 4 Threads



(d) LLC Behavior of SVM With 4 Threads

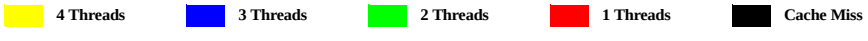


Figure 5: Time Varying Behavior of Parallel Bioinformatics Workloads. The figure shows the time varying behavior of those benchmarks that exhibited data sharing (SNP exhibited no sharing hence is not displayed). The x-axis represents the total number of instructions executed (in billions) and the y-axis represents the appropriate metric presented as a percent. The private last level cache miss-rate is presented as the average miss-rate of all private caches. This figure is best viewed as a soft copy or a color printout.